title: TypeScript SDK description: An overview of the TypeScript SDK (suave-viem)

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

:::info

SUAVE-Viem is a fork of viem that will eventually be upstreamed but is currently still in a dynamic state.

Sending Confidential Compute Requests works slightly differently, but most other functionality is similar to interacting with any other EVM chain from viem.

:::

This page describes how to work with the SUAVE-viem TypeScript SDK. The SDK simplifies interaction with the SUAVE Chain and provides easy-to-use functions to send transactions and query data. Below, you'll find steps on how to install the library and perform some basic actions.

Installation

The @flashbots/suave-viem package is available on NPM, and can be installed with any NPM-based package manager, such as **npm**, **yarn**, or **bun**.

bash npm i @flashbots/suave-viem bash yarn add @flashbots/suave-viem bash bun add @flashbots/suave-viem

Instantiation

:::info

The rest of this guide assumes you have SUAVE running locally.

:::

First, you need to import necessary modules and instantiate the client. In your index.ts file, you can copy and paste the following:

```typescript import {http} from '@flashbots/suave-viem'; import {getSuaveProvider} from '@flashbots/suave-viem/chains/utils';

// connect to your local SUAVE node const SUAVE_RPC_URL = 'http://localhost:8545'; const suaveProvider = getSuaveProvider(http(SUAVE_RPC_URL)); ```

## Wallet Creation

To interact with the SUAVE network, we'll first need a wallet. When running SUAVE locally, there is an account which is set up with funds for you by default. Paste the following the following block to instantiate it in viem:

```typescript // plus other imports from above import {Hex} from '@flashbots/suave-viem'; import {getSuaveWallet} from '@flashbots/suave-viem/chains/utils';

const DEFAULT_PRIVATE_KEY: Hex = '0x91ab9a7e53c220e6210460b65a7a3bb2ca181412a8a7b43ff336b3df1737ce12';

const defaultWallet = getSuaveWallet({ transport: http(SUAVE_RPC_URL), privateKey: DEFAULT_PRIVATE_KEY, });

console.log('Wallet Address:', defaultWallet.account.address); ```

You can now run this file:

bash bun run index.ts

And you should see the following printed to your terminal:

bash Wallet Address: 0xBE69d72ca5f88aCba033a063dF5DBe43a4148De0

## Watching Pending Transactions

You can watch for pending transactions and log their details using the following example:

typescript // Watch for pending transactions suaveProvider.watchPendingTransactions({ async onTransactions(transactions) { for (const hash of transactions) { try { const receipt = await suaveProvider.getTransactionReceipt({hash}); console.log('Transaction Receipt:', receipt); } catch (error) { console.error('Error fetching receipt:', error); } } }, });

## Send a Confidential Compute Request

Let's walk through how to set up and send a Confidential Compute Request:

#### 1. Get Current Gas Price

First, instantiate a new wallet of your own, and fetch the current gas price from the network.

```typescript const PRIVATE_KEY: Hex = ';

const wallet = getSuaveWallet({ transport: http(SUAVE_RPC_URL), privateKey: DEFAULT_PRIVATE_KEY, }); ```

#### 2. Prepare the Fund Transaction

Create a transaction object to fund your new wallet with the required amount.

```typescript import {TransactionRequestSuave} from '@flashbots/suave-viem/chains/suave/types'; // ...

const fundTx: TransactionRequestSuave = { type: '0x0', value: 100000000000000001n, gasPrice: 10000000000n, // 10 gwei is typically fine for testing to: wallet.account.address, gas: 21000n, }; ```

#### 3. Send the Fund Transaction and wait for confirmation

Send the transaction to fund the wallet.

typescript const fund = await defaultWallet.sendTransaction(fundTx); console.log('sent fund tx', fund);

Use a while loop to periodically check if the transaction has been confirmed.

typescript while (true) { const fundReceipt = await suaveProvider.getTransactionReceipt({ hash: fund, }); if (fundReceipt) { console.log('fund tx landed', fundReceipt); break; } await sleep(4000); }

If you once again run bun run index.ts, you should see something like the following logged to your terminal:

txt sent fund tx 0xe6f7385c6992c91941fe10e43feaf987cb4377f302915945b747fcb0e0a7f40a formatting tx receipt fund tx landed { blockHash: "0x23c08d5a680b75ca10f4e0d737eadade2ec4c6eb66dae6edea88867033bd436e", blockNumber: 2n, contractAddress: null, cumulativeGasUsed: 21000n, effectiveGasPrice: 3000000001n, from: "0xbe69d72ca5f88acba033a063df5dbe43a4148de0", gasUsed: 21000n, logs: [], logsBloom: "0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 status: "success", to: "0x5f3371793285920351344a1eaaaa48d45e600652", transactionHash: "0xe6f7385c6992c91941fe10e43feaf987cb4377f302915945b747fcb0e0a7f40a", transactionIndex: 0, type: "legacy" }

#### 4. Create a Confidential Compute Request

Now, let's set up a CCR with the appropriate parameters.

```typescript import { TransactionRequestSuave, SuaveTxRequestTypes, } from '@flashbots/suave-viem/chains/utils'; // ...

const ccr: TransactionRequestSuave = { confidentialInputs: '0x00000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000000000000000000000fd7b22626c6f636b4e756d626572223a2 kettleAddress: '0xB5fEAfbDD752ad52Afb7e1bD2E40432A485bBB7F', // Address of your local Kettle. Use 0x03493869959C866713C33669cA118E774A30A0E5 if on Rigil. to: '0x8f21Fdd6B4f4CacD33151777A46c122797c8BF17', gasPrice: 10000000000n, // Gas price for the transaction gas: 420000n, // Gas limit for the transaction type: SuaveTxRequestTypes.ConfidentialRequest, // SUAVE transaction request type ("0x43") chainId: suaveRigil.id, data:
```

'0x236eb5a700000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000000000000000000600000000000000000000000
// Data payload for the transaction }; ```

:::info

confidentialInputs is a field to store information that should be kept private during computation, and the data field is the typical calldata required to interact with a dapp.

If you prefer TypeScript, you can see how to craft your own CCRs in theexamples directory of suave-viem.

:::

5. Send the Confidential Compute Request

Finally, send the CCR to the network.

typescript const res = await wallet.sendTransaction(ccr); console.log(`sent ccr! tx hash: ${res}`);

You should see the transaction hash logged to your terminal, like this:

bash sent ccr! tx hash: 0xad488fd0a2b428bfa30c7ef8f8ce12e2f7f2554643ad1ca94d15ab11ad5dd9dd

Fetching Blockchain Data

To fetch the latest block or transaction receipt, you can use the following functions:

```typescript async function fetchBlockchainData() { // Get the number of the latest block const latestBlockNumber = await suaveProvider.getBlockNumber();

console.log('Block number: ', latestBlockNumber);

// Fetch the latest block const latestBlock = await suaveProvider.getBlock({ blockNumber: latestBlockNumber, includeTransactions: false, });

console.log('Latest Block:', latestBlock); }

fetchBlockchainData(); ```
```